算法作业及参考答案

■ 作业一

1. 对于下列各组函数$f(n)$和$g(n)$, 确定$f(n)=O\big(g(n)\big)$或$f(n)=\Omega\big(g(n)\big)$或$f(n)=\theta\big(g(n)\big)$, 并简述理由。

(1) $f(n)=\lg n^2;\ g(n)=\lg n+5$
(2) $f(n)=\lg n^2;\ g(n)=\sqrt{n}$
(3) $f(n)=n;\ g(n)=\lg^2 n$
(4) $f(n)=n\lg n+n;\ g(n)=\lg n$
(5) $f(n)=10;\ g(n)=\lg 10$
(6) $f(n)=\lg^2 n;\ g(n)=\lg n$
(7) $f(n)=2^n;\ g(n)=100n^2$
(8) $f(n)=2^n;\ g(n)=3^n$

解：

(1) $\lg n^2 = \theta(\lg n+5)$
(2) $\lg n^2 = O(\sqrt{n})$
(3) $n=\Omega(\lg^2 n)$
(4) $n\lg n+n=\Omega(\lg n)$
(5) $10=\theta(\lg 10)$
(6) $\lg^2 n=\Omega(\lg n)$
(7) $2^n=\Omega(100n^2)$
(8) $2^n=O(3^n)$

2．证明：$n!=\mathrm{o}(n^n)$

证明：由 stirling 公式得，$n!=\sqrt{2\pi n}\left(\frac{n}{e}\right)^{\mathrm{n}}\left[1+\theta\left(\frac{1}{n}\right)\right]$，

$$\lim_{n\to\infty}\frac{n!}{n^n}=\lim_{n\to\infty}\frac{\sqrt{2\pi n}\left[1+\theta\left(\frac{1}{n}\right)\right]}{e^n}=0$$

3．求下列函数的渐近表达式:

$3n^2+10n;\ n^2/10+2^n;\ 21+1/n;\ \lg n^3;\ 10\lg 3^n$。

解：

$3n^2+10n=O(n^2)$
$n^2/10+2^n=O(2^n)$
$21+1/n=O(1)$
$\lg n^3=O(\lg n)$
$10\lg 3^n=O(n)$

4．假设$f(n)$与$g(n)$都是渐近非负函数。使用θ记号的基本定义来证明$\max\big(f(n),g(n)\big)=\theta\big(f(n)+g(n)\big)$

证明：因为$f(n)$与$g(n)$都是渐近非负函数，根据定义有：存在N_f,N_g使得：当$n>N_f$时，$f(n)\geq 0$，同时，当$n>N_g$时，$g(n)\geq 0$。所以，我们取$N_0=max\big(N_f,N_g\big)$，此时，当$n>N_0$时，同时有$f(n)\geq 0,g(n)\geq 0$。取$C_1=1/2,C_2=1$，则当$n>N_0$时，有：

$$(f(n)+g(n))/2\ \leq max(f(n),g(n))\ \leq\ f(n)+g(n)$$

得证。

5．$2^{n+1}=O(2^n)$成立吗？$2^{2n}=O(2^n)$成立吗?

解：1）$2^{n+1}=O(2^n)$成立

取$c \geq 2$，当$n > 0$时，

有$2^{n+1} = 2 * 2^n \leq c * 2^n$,

因此，成立

2）$2^{2n} = O(2^n)$不成立

如果$2^{2n} = O(2^n)$成立，根据定义有：$2^{2n} \leq c2^n$（c为常数）

得$n \leq \lg c$，则$2^{2n} \leq c2^n$对于任意大的n不成立，与假设矛盾。

因此，$2^{2n} = O(2^n)$不成立

■ 作业二

1. 使用 Strassen 算法计算如下矩阵乘法：

$$\begin{bmatrix}1 & 3\\ 7 & 5\end{bmatrix}\begin{bmatrix}6 & 8\\ 4 & 2\end{bmatrix}$$

给出计算过程。

解：

$A_{11}=1, A_{12}=3, A_{21}=7, A_{22}=5$
$B_{11}=6, B_{12}=8, B_{21}=4, B_{22}=2$
$S_1=B_{12}-B_{22}=8-2=6$
$S_2=A_{11}+A_{12}=1+3=4$
$S_3=A_{21}+A_{22}=7+5=12$
$S_4=B_{21}-B_{11}=4-6=-2$
$S_5=A_{11}+A_{22}=1+5=6$
$S_6=B_{11}+B_{22}=6+2=8$
$S_7=A_{12}-A_{22}=3-5=-2$
$S_8=B_{21}+B_{22}=4+2=6$
$S_9=A_{11}-A_{21}=1-7=-6$
$S_{10}=B_{11}+B_{12}=6+8=14$
$P_1=A_{11}S_1=1*6=6$
$P_2=S_2B_{22}=4*2=8$
$P_3=S_3B_{11}=12*6=72$
$P_4=A_{22}S_4=5*(-2)=-10$
$P_5=S_5S_6=6*8=48$
$P_6=S_7S_8=-2*6=-12$
$P_7=S_9S_{10}=-6*14=-84$
$C_{11}=P_5+P_4-P_2+P_6=48+(-10)-8+(-12)=18$
$C_{12}=P_1+P_2=6+8=14$
$C_{21}=P_3+P_4=62$
$C_{22}=P_5+P_1-P_3-P_7=48+6-72-(-84)=66$

2. 证明：$T(n)=2T(\lfloor n/2\rfloor+17)+n$的解为$O(n\lg n)$。

往证: 存在常数 N_0, C_0, 使得: 当 $n>N_0$ 时, $T(n)\le C_0 n\lg n$ 。

假设: $T(\lfloor n/2\rfloor+17)\le C_0(\lfloor n/2\rfloor+17)\lg(\lfloor n/2\rfloor+17)$

有：
$$\begin{aligned}T(n)&\le 2C_0\left(\lfloor n/2\rfloor+17\right)\lg\left(\lfloor n/2\rfloor+17\right)+\mathrm{n}\\&\le 2C_0\left(n/2+17\right)\lg\left(n/2+17\right)+\mathrm{n}\\&=C_0(n+34)[\lg(n+34)-1]+\mathrm{n}\\&=C_0 n\lg(n+34)-C_0 n+34C_0\lg(n+34)-34C_0+\mathrm{n}\qquad(*1)\end{aligned}$$

取$C_0\ge 2$,

则有
$$\begin{aligned}(*1)&\le C_0 n\lg(n+34)+34C_0\lg(n+34)-34C_0-\mathrm{n}\\&<C_0 n\lg(n+34)+34C_0\lg(n+34)-\mathrm{n}\\&=C_0 n\lg n+C_0 n\lg(n+34)-C_0 n\lg n+34C_0\lg(n+34)-\mathrm{n}\quad(*2)\end{aligned}$$

下面我们只需证明存在常数 $N_0, C_0\ge 2$, 使得当 $n>N_0$ 时，有

$$C_0 n\lg(n+34)-C_0 n\lg n+34C_0\lg(n+34)-\mathrm{n}\le 0$$

即有:

$$(*2) \le C_0 n\lg n$$
$$C_0 n\lg(n+34) - C_0 n\lg n + 34C_0\lg(n+34) - n$$
$$= C_0 n(\lg(n+34) - \lg n) + 34C_0\lg(n+34) - n$$
$$= C_0 n(\ln(1+34/n)/\ln 2) + 34C_0\lg(n+34) - n$$
$$\le C_0 n(34/(n\ln 2)) + 34C_0\lg(n+34) - n$$

此时只需取 $C_0 = 3, N_0 = 100$, 即可使当 $n > N_0$ 时, 有

$$C_0 n\lg(n+34) - C_0 n\lg n + 34C_0\lg(n+34) - n \le 0$$

原题得证

3. 使用 4.5 节中的主方法，可以证明 $T(n) = 4T(n/3) + n$ 的解为$T(n) = \Theta(n^{\log_3 4})$。说明基于假设 $T(n) \le cn^{\log_3 4}$的代入法不能证明这一结论。然后说明如何通过减去一个低阶项完成代入法证明。

解：

假设$T(n) \le cn^{\log_3 4}$，

有$T(n) \le 4c(n/3)^{\log_3 4} + n = cn^{\log_3 4} + n$,

但这并不意味着对于任意 c 都有$T(n) \le cn^{\log_3 4}$成立。

因此考虑减去一个低阶项，猜测$T(n) \le cn^{\log_3 4} - dn$，$d$为大于 0 的常数。

此时有$T(n) \le 4\left(c(n/3)^{\log_3 4} - dn/3\right) + n$

$$= cn^{\log_3 4} - 4dn/3 + n$$
$$\le cn^{\log_3 4} - dn$$

只要$d \ge 3$，上式即可成立。

4. 分治法求最大子数组

问题背景

	1	2	3	4	5	6	7	8	9	10
X	-1	-3	3	5	-4	3	2	-2	3	6

子数组：数组中连续的一段序列，例如 X[3..7]；

子数组和：子数组中元素的和，X[3..7]的和就是 3+5-4+3+2=9。

问题定义

输入：

给定一个数组 X[1..n]，对于任意一对数组下标为 l, r (l ≤ r)的非空子数组，其和记为 S(l, r) = $\sum_{i=l}^{r} X[i]$。

输出：

S(l, r)的最大值，记为S_{max}。

解：

分解原问题：将数组 X[1..n]分为 X[1..n/2]和 X[n/2+1..n];

解决子问题：

- S_1：数组 X[1..n/2]的最大子数组；
- S_2：数组 X[n/2+1..n]的最大子数组；

合并问题：

- S_3：跨中点的最大子数组。

- 数组 X 的最大子数组之和 $S_{max} = \max\{S_1, S_2, S_3\}$

子问题 S_1 和 S_2 可通过递归的方式求解，关键在于合并时 S_3 如何求解？
用 mid 表示中点，那么跨越中点的最大子数组必定包括 X[mid]和 X[mid+1]，可使用 S_{left} 记录以 X[mid]结尾的最大子数组和，S_{right} 记录以 X[mid+1]开始的最大子数组和。则 $S_3 = S_{left} + S_{right}$。
求 left 可以从 X[mid]向前遍历，求 right 可以从 X[mid+1]向后遍历。因此求解 S_3 的时间复杂度是 O(n)。

伪代码：

```
1.  int maxSubArray(int[] X, int low, int high){
2.      if (low == high)
3.          return X[low]
4.
5.      mid = (low + high) // 2
6.      S1 = maSubArray(X, low, mid)
7.      S2 = maSubArray(X, mid+1, high)
8.      S3 = crossingSubArrat(X, low, mid, high)
9.      return max(S1, S2, S3)
10. }
11.
12. int maxSubArray(int[] X, int low, int mid, int high){
13.     S_left = -inf  # 初始化为无穷小
14.     S_right = -inf
15.
16.     sum = 0
17.     for(i=mid; i>=low; i--){
18.         sum = sum + X[i]
19.         S_left = max(S_left, sum)
20.     }
21.
22.     sum = 0
23.     for(i=mid+1; i<=high; i++){
24.         sum = sum + X[i]
25.         S_right = max(S_right, sum)
26.     }
27.
28.     return S_left + S_right
29. }
```

调用 maxSubArra(X, 1, n)即可。

时间复杂度分析：

$$T(n)=\begin{cases}1, & n=1\\ 2T\left(\frac{n}{2}\right)+O(n), & n>1\end{cases}$$

由主定理可以求得$T(n)=n\log n$。

5. 计数逆序问题

问题背景

逆序对：当 i < j 时，A[i] > A[j]的二元组(A[i], A[j])。例如下图中的(A[1], A[4])、(A[2], A[4])等。

	1	2	3	4	5
A	4	6	8	3	5

问题定义

输入：

长度为 n 的数组 A[1..n]。

输出：

数组 A[1..n]逆序对的总数，即$\sum_{1\le i<j\le n}X_{i,j}$，其中

$$X_{i,j}=\begin{cases}1, & A[i]>A[j]\\ 0, & A[i]\le A[j]\end{cases}$$

解：

分解原问题：把数组 A[1..n]二分为两个子数组 A[1.. n/2]和 A[n/2+1..n];

递归求解子问题：

- S_1：A[1.. n/2]中的逆序对数组；
- S_2：A[n/2+1..n]中的逆序对数；

合并问题：

- S_3：跨越不同子数组的逆序对数。
- 数组 A 的逆序对数 $S = S_1 + S_2 + S_3$

问题在于 S_3 如何求解？

考虑归并排序时利用子数组的有序性完成逆序数的计算：具体而言，对于 $i \le mid, j > mid$，且 A[i] > A[j]时，A[i..mid]中的元素均与 A[j]构成逆序对，因此通过一次线性扫描就可以完成所有右子数组元素在左数组中的定位。

- 从左到右扫描有序子数组：$A[i] \in A[1..m], A[j] \in A[m+1..n]$;
  - 如果 $A[i] > A[j]$，统计逆序数，j 向右移；
  - 如果 $A[i] \le A[j]$，i 向右移。
- 利用归并排序框架保证合并后数组的有序性

伪代码：

```
1.  int countInver(int[] A, int left, int right){
2.      if (left >= right)
```

```
        return 0

    mid = (left + right) // 2
    S1 = countInver(A, left, mid)
    S2 = countInver(A, mid+1, right)
    S3 = mergeCount(A, left, mid, right)
    S = S1 + S2 + S3
    return S
}

int mergeCount(int[] A, int left, int mid, int right){
    temp = copy(A)  # 使用 temp 临时存储 A, 以便完成归并排序
    S3 = 0
    i=left, j=mid+1, k=0
    while(i<=mid && j<=right){
        if (temp[i] <= temp[j]){  # 不构成逆序对
            A[left+k] = temp[i]
            k++, i++
        }
        else{
            A[left+k] = temp[j]
            k++, j++
            S3 = S3 + (mid - i + 1)
        }
    }

    while(i <= mid){
        A[k++] = temp[i++]
    }
    while(j <= right){
        A[k++] = temp[j++]
    }

    return S3
}
```

调用 countInver(A, 1, n)即可。

时间复杂度分析：

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n), & n > 1 \end{cases}$$

由主定理可以求得$T(n) = n \log n$。

■ 作业三

1. 对矩阵规模序列<5，10，3，12，5，50，6>，求矩阵链最优括号化方案。

解：由题意得

矩阵	A_1	A_2	A_3	A_4	A_5	A_6
规模	5 × 10	10 × 3	3× 12	12 × 5	5 × 50	50 × 6

其中$< p_0, p_1, p_2, p_3, p_4, p_5, p_6 >=< 5,10,3,12,5,50,6 >$

利用公式计算m表：

$$m[i,j] = \begin{cases} 0 & \text{如果 } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{如果 } i < j \end{cases}$$

计算得下表：

i\j	**1**	**2**	**3**	**4**	**5**	**6**
1	0	150	330	405	1655	2010
2		0	360	330	2430	1950
3			0	180	930	1770
4				0	3000	1860
5					0	1500
6						0

相应的s表为：

i\j	**2**	**3**	**4**	**5**	**6**
1	1	2	2	4	2
2		2	3	2	2
3			3	4	4
4				4	4
5					5

易得，矩阵链最优括号化方案为$((A_1A_2)((A_3A_4)(A_5A_6)))$。

2. 用代入法证明递归公式(15.6)的结果为$\Omega(2^n)$。

证明：递归公式(15.6)如下：

$$P(n) = \begin{cases} 1 & \text{如果} n = 1 \\ \sum\limits_{k=1}^{n-1} P(k)P(n-k) & \text{如果} n \ge 2 \end{cases}$$

即证，$P(n) = \Omega(2^n)$，亦即证，存在正常数c和n_0，当$n \ge n_0$时，使$P(n) \ge c2^n$成立。当$n = 1$时，有$P(n) = 1$，成立。

假设当$n < m$时，有$P(n) \ge c2^n$成立。

则当$n = m$时，有

$$\begin{aligned} P(m) &\ge \sum_{k=1}^{m-1} P(k)P(m-k) \\ &\ge \sum_{k=1}^{m-1} c2^k c2^{m-k} \\ &\ge c^2(m-1)2^m \end{aligned}$$

取$n_0 = \frac{1}{c} + 1$，当$n \ge n_0$时，有$P(m) \ge c2^m$。

证毕。

3. 考虑矩阵链乘法问题的一个变形：目标改为最大化矩阵序列括号化方案的标量乘法运算次数，而非最小化。此问题具有最优子结构性质吗？

解：这个问题具有最优子结构。

分析如下：

采用动态规划法的第一步就是寻找最优子结构，然后利用这一子结构，就可以根据子问题的最优解构造出原问题的一个最优解。

用记号$A_{i\cdots j}$表示对乘积$A_iA_{i+1}\cdots A_j$的求积结果，其中$i<j$。

这个问题的最优子结构如下：假设$A_iA_{i+1}\cdots A_j$的一个最优括号化方案把A_k与A_{k+1}之间分开，则对$A_iA_{i+1}\cdots A_j$最优括号化方案的“前缀”子链$A_iA_{i+1}\cdots A_k$的括号化方案必须是$A_iA_{i+1}\cdots A_k$的一个最优括号化方案，因为如果对括号化方案$A_iA_{i+1}\cdots A_k$有一个代价更大的括号化方案，那么把它替换到$A_iA_{i+1}\cdots A_j$的最优括号化方案中就会产生$A_iA_{i+1}\cdots A_j$的另一种括号化方案，而它会具有更大的代价，产生矛盾。

所以，判断这个问题具有最优子结构。

4. 设计 LCS-LENGTH 的带备忘的版本，运行时间为$O(mn)$。

解：假设 X、Y 为需要求解最长公共子序列的字符串，$c[i,j]$为 X 中前i个字符，Y 中前j个字符的最长公共子序列的长度。

利用如下公式设计算法：

$$c[i,j]=\begin{cases}0 & \text{if } i=0 \text{ or } j=0\\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i=y_i\\ max\{c[i,j-1],c[i-1,j]\} & \text{if } i,j>0 \text{ and } x_i\neq y_i\end{cases}$$

```
int c[1005][1005]; //存储 X 中前 i 个字符，Y 中前 j 个字符的最长公共子序列的长度

int LOOKUP_LENGTH(char * X, char * Y, int i, int j)
//求解 c[i][j]
{
    if (c[i][j] > -1) {
        return c[i][j];
    }
    if (i == 0 || j == 0) {
        c[i][j] = 0;
    } else {
        if (X[i - 1] == Y[j - 1]) //X 中第 i 个字符与 Y 中第 j 个字符相等
        {
            c[i][j] = LOOKUP_LENGTH(X, Y, i - 1, j - 1) + 1;
        } else //X 中第 i 个字符与 Y 中第 j 个字符不相等
        {
            LOOKUP_LENGTH(X, Y, i, j - 1);
            LOOKUP_LENGTH(X, Y, i - 1, j);
            c[i][j] = c[i][j - 1] > c[i - 1][j] ? c[i][j - 1] : c[i - 1][j];
        }
    }
```

```
    return c[i][j];
}

int LCS_LENGTH(char * X, char * Y, int m, int n)
//求解 X、Y 的最长公共子序列的长度
{
    int i, j;
    for (i = 0; i <= m; i++) {
        for (j = 0; j <= n; j++) {
            c[i][j] = -1; //初始化
        }
    }
    return LOOKUP_LENGTH(X, Y, m, n);
}
```

5. 设计一个$O(n^2)$时间的算法，求一个n个数的序列的最长单调递增子序列。

解：假设序列为x，$c[i]$为以$x[i]$结尾的单调递增子序列的长度，$pre[i]$为以$x[i]$结尾的单调递增子序列中$x[i]$前一个字符的位置

利用如下公式设计算法：

$$c[i] = max\{\max_{1\le j<i\&x[i]>x[j]}\{c[j]\}+1, 1\}$$

$$pre[i] = \begin{cases} 0 & \text{if } c[i]=1 \\ j & \text{if } c[i]>1 \text{ and } x[j] \text{ is the previous number of } x[i] \end{cases}$$

```
int c[1005], pre[1005], longest_subsequence[1005];
//longest_subsequence 存储最长单调递增子序列

void longest_monotonically_increasing_subsequence(int *x, int n)
//求解最长单调递增子序列
{
    int i, j;
    for (i = 1; i <= n; i++) {
        c[i] = 1;
        pre[i] = 0;
        for (j = 1; j < i; j++) {
            if (x[i] > x[j] && c[i] < c[j] + 1) {
                c[i] = c[j] + 1;
                pre[i] = j;
            }
        }
    }

    int max = c[1];
    int max_index = 1;
```

```
    for (i = 2; i <= n; i++) {
        if (max < c[i]) {
            max = c[i];
            max_index = i;
        }
    }
    j = 0;
    while (pre[max_index]) {
        longest_subsequence[j++] = max_index;
        max_index = pre[max_index];
    }
    longest_subsequence[j++] = max_index;

    printf("最长单调递增子序列的长度为：%d\n", j);
    printf("最长单调递增子序列为：\n");
    for (i = j - 1; i >= 0; i--) {
        printf("%d ", longest_subsequence[i]);
    }
    printf("\n");
}
```